REMARKS

Claims 1-21 are pending. Claims 1, 14, 17, and 20 are independent.

Applicant amended independent claim 1 for greater clarity.

The examiner rejected claims 1-3, 6, 10, 13-21 under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 4,724,521 to Carron et al, in view of U.S. Patent No. 5,802,373 to Yates and U.S. Patent No. 5,748,950 to White et al.

The Examiner also rejected Claims 4 and 5 under 35 U.S.C. 103(a) as being unpatentable over Carron, White and Yates, in view of U.S. Patent No. 5,898,866 to Atkins et al.

The Examiner also rejected claims 7, 8, and 11 under 35 U.S.C. § 103(a) as being unpatentable over Carron, White and Yates, in view of U.S. Patent No. 4,742,151 to Bruckert et al.

The Examiner further rejected claim 9 under 35 U.S.C. § 103(a) as being unpatentable over Carron, White and Yates, in view of U.S. Patent No. 5,202,972 Gusefski et al.

The Examiner additionally rejected claim 12 under 35 U.S.C. § 103(a) as being unpatentable over Carron, White and Yates, in view of U.S. Patent No. 6,139,199 to Rodriguez.

Independent claim 1 recites, "executing a branch instruction that causes a processor to branch from executing a first sequential series of instructions to a different sequential series of instructions based on a byte, specified by the branch instruction, in a register, specified by the branch instruction, being equal or not equal to a byte value specified by the branch instruction."

With respect to independent claim 1, the examiner admitted that "Carron does not teach a branch instruction that causes a processor to compare and branch based on a byte specified by the branch instruction; and of comparing a byte specified by the branch in instruction in a register and performing branching instructions based on the comparison." (Office Action, page 3, paragraph 5). The examiner contends that White and Yates teach these features. Applicant disagrees.

White describes an optimized compare-and branch (COBR) instruction for execution in a RISC-type processor (Abstract). As shown in FIG. 3, White's COBR instruction includes the opcode, the operands SRC1 and SRC2, and the displacement field (i.e., the displacement to the target branching address). White explains that:

> **When a COBR instruction is executed, the microprocessor compares the source 2 and source 1 operands provided with the instruction and sets a condition code in a register located in the instruction sequencer according to the comparison**

results. Subsequently, a portion of the instruction's opcode is compared against the actual condition code. If the comparison results in a match, then the processor branches to an instruction specified by the displacement value propagated with the COBR instruction. Otherwise, the processor goes to the next sequential instruction. Mechanisms for comparing condition codes to a portion of an instruction's opcode are well known for determining branch decisions and will not be described more fully herein. (Col. 5, lines 44-56)

With respect to the operands used by White's various instructions, including the COBR instruction, White further explains:

Within the processor illustrated in FIG. 2, all operations take place at the register level. Source operands specify either a global register, a local register or a constant value as instruction operands. The functional units are coupled to the register file 30 via three independent 32-bit buses. These are identified as source 1 (SRC1), source 2 (SRC2) and the destination (DEST) buses. In alternative embodiments of the present invention, a wider single bus, or smaller multiplexed common bus may be utilized (or various combinations of separate buses) for communicating between the register file 30 and the various functional units. (col. 5, lines 14-24)

Thus, White's COBR instruction can perform a comparison operation using registers and/or constant values. However, White's COBR instruction does not specify, and White neither describes nor suggests a particular byte of a particular register as the value that is to be compared to a specified value. Rather, White's COBR instruction compares an entire register to another entire register or to a constant value. Accordingly, White neither discloses nor suggests at least the feature of "executing a branch instruction that causes a processor to branch from executing a first sequential series of instructions to a different sequential series of instructions based on a byte, specified by the branch instruction, in a register, specified by the branch instruction, being equal or not equal to a byte value specified by the branch instruction," as required by applicant's independent claim 1.

As for Yates, as explained in applicant's Amendment in Reply to Action of October 4, 2005 (hereinafter the "previous Amendment in Reply"), Yates describes a computer system for executing a binary image conversion which converts instructions from an instruction set of a first non-native system to a second native computer system (Abstract). Amongst the operations that Yates' system performs is condition code processing in transformer (described at col. 76, line 11 to col. 78, line 34). One such condition code processing sequence is shown in Yates' FIG. 65B. As explained by Yates, "[s]ource instruction 884a performs a byte compare of register AL to the constant 3.

Instruction 884b performs a branch if the value contained in the register AL is not equal to 3" (col. 77, lines 29-32).

Thus, in contrast to applicant's claimed method, Yates requires two separate instructions to perform the branching operation, namely, a compare instruction, followed by a separate conditional branch instruction. And although the compare instruction 884a is referred to as byte compare instruction, both FIG. 65B and the specification fail to disclose that a particular specified byte of the AL register is compared to the value 3. Rather, Yates states that "[i]nstruction 884b performs a branch if the value contained in the register AL is not equal to 3" (col. 77, lines 31-32). In other words, Yates branching decision is based on the entire value of the AL register being equal or not equal to 3. Accordingly, Yates also does not disclose or suggests at least the feature of "executing a branch instruction that causes a processor to branch from executing a first sequential series of instructions to a different sequential series of instructions based on a byte, specified by the branch instruction, in a register, specified by the branch instruction, being equal or not equal to a byte value specified by the branch instruction," as required by applicant's independent claim 1.

Thus, because none of the cited references discloses or suggests, alone or in combination, at least the feature of: "executing a branch instruction that causes a processor to branch from executing a first sequential series of instructions to a different sequential series of instructions based on a byte, specified by the branch instruction, in a register, specified by the branch instruction, being equal or not equal to a byte value specified by the branch instruction," applicant's independent claim 1 is patentable over the cited art.

Moreover, even assuming, *arguendo*, that one of White and/or Yates taught the above features, applicant contends that the examiner has failed to show the necessary motivation to combine White and/or Yates with Caron to establish a *prima facia* case of obviousness with respect to independent claim 1.

As explained in applicant's previous Amendment in Reply, Carron describes a method for operating a local terminal that includes executing pre-arranged application programs. Specifically, Carron explains:

> **the method of this invention is based on storing in read only memory circuits within the local terminal a number of general purpose operation routines which**

comprise instructions to be executed by the central processor unit to accomplish a particular program task. Each of these general purpose operation routines is associated with a defined command which, in its object code version, includes an operation code. The object code version of the commands associated with the general purpose operation routines has a code length substantially shorter than the code length of the operation routine. In accordance with the method of this invention, the local terminal also has a program established in its read only memory system for interpreting the operation code to access the associated general purpose operation routine for execution by the central processor unit. The object code version of the application program in the form of a sequence of commands has a code size which is several times smaller than the compiled code size would be for an entire application program which could be directly executed after downloading.

More specifically, the method of this invention includes a first step of establishing a set of general purpose operation routines to be executed by the local computer system. Each of these general purpose operation routines comprises a set of instructions for execution by the central processor unit in a prearranged manner to accomplish a specific task. The next step is to store the set of general purpose operation routines in the read only memory of the local terminal so that they may be accessed for execution by the central processor unit. The storage locations of these general purpose operation routines will be arranged and noted so that the routines can be addressed.

A following step is to define a set of commands, each of which is associated with a specific one of the general purpose operation routines. Each of the commands includes at least an operation code relating the command to its associated general purpose operation routine. Each of the commands is defined such that it has an associated command code length substantially less than the code length of the associated general purpose operation routine. Preferably, for convenience of writing programs, each command is defined with a high level programming syntax in which the command is represented by a series of word forms in abbreviated notation which have a recognizable association with the task that the associated general purpose operation routine will perform when it is executed by the central processor unit in the local terminal. This high level form of the command is then compiled and assembled to produce the operation code which relates to the associated general purpose operation routine in a manner which is intelligible to the interpreter program in the local terminal. (emphasis added, col. 7, line 59, to col. 8, line 47).

Thus, Carron defines commands that correspond to routines, stored in memory, each of which comprises several processor-executable instructions. For example, one of these commands, listed in Carron's "SECTION FOUR: ALPHABETIC LISTING OF COMMANDS", is the COMP_BYTE command. Carron's COMP_BYTE, much like Carron's other commands, is not a single processor-executable instruction, but rather a command that invokes a routine comprising several instructions stored in memory. Invoking the COMP_BYTE command routine causes a byte at a specified position in a designated buffer to be compared with a byte constant in the command, and if the test passes (presumably, if the values of the compared operands match) the COMP_BYTE command routine causes a branching operation to be performed (col. 134, lines 30-34).

Carron further explains that one of the advantages of its programming methodology is that:

> **It should be apparent from the above description that the method of this invention makes it possible for the programmer to create application programs with commands in a source level language that are particularly suited to the type of application for the terminals that are to be programmed. The commands are compiled into compressed operation codes and parameters which reduce the size of the code which must be downloaded to a practicable size. The methods of this invention provide for parallel execution of APMs which greatly expands the facility with which the programmer can implement the execution of the program tasks in the local terminal without wasting time for inputs to arrive. This increases the efficiency of use of the central processor unit within the terminal. (Carron, col. 105, lines 49-63)**

Thus, Carron's commands correspond to a high-level programming language that a programmer may use to prepare programs that include sets of commands that can be executed on different processors. Each processor on which the set of program commands is executed will have corresponding routines associated with that processor's architecture and stored on a corresponding storage module. Accordingly, commands, such as Carron's COMP_BYTE command, cannot include instructions that are unique to a particular processor because those commands could not then be executed on some other processor architecture.

White's COBR instruction is a machine level instruction configured to be executed in the particular processing environment shown in FIG. 2 (see White, col. 5, lines 14-42). As for Yates instruction sequences, Yates converts received non-native instruction sequences into instruction sequences that can be executed on the native microprocessor. Thus, Yates instruction sequences, including the instruction sequences 884a and 884b shown in FIG. 65B, are also machine-level instructions configured for execution in a particular processing environment.

Accordingly, to modify Carron's high-level commands by combining it with White's and/or Yates machine-level instructions would render Carron unsatisfactory for its intended purpose and/or change Carron's principle of operation. Indeed, not only are Carron's high-level commands incompatible with White's and Yates' machine-level instruction, but also use of White's and Yates processor-specific machine-instruction would preclude Carron's commands from executing on different types of processors.

Because combining White and/or Yates with Carron would render Carron unsatisfactory for its intended purpose and/or would change Carron's principle of operation, no motivation for combining Carron with Yates and/or White exists. The examiner has therefore failed to establish a *prima facie* case of obviousness with respect to independent claim 1. Independent claim 1 is thus patentable over the cited art.

Claims 2-13 depend from independent claim 1. Accordingly, claims 2-13 are patentable for at least the same reasons as independent claim 1.

Independent claims 14, 17 and 20 recite "a branch instruction that causes a processor to: fetch a byte, specified by the branch instruction, stored in a register, specified by the branch instruction; determine whether the byte in the register is equal or not equal to a specified byte value contained in the branch instruction; and perform a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register." For reasons similar to those provided with respect to independent claim 1, independent claims 14, 17, and 20 are patentable over the cited art.

Claims 15-16 depend from independent claim 14 and are therefore patentable for at least the same reasons as independent claim 14. Claims 18-19 depend from independent claim 17 and are therefore patentable for at least the same reasons as independent claim 17. Claim 21 depends from independent claim 20 and is therefore patentable for at least the same reasons as independent claim 20.

It is believed that all the rejections and/or objections raised by the examiner have been addressed.

In view of the foregoing, applicant respectfully submits that the application is in condition for allowance and such action is respectfully requested at the Examiner's earliest convenience.

All of the dependent claims are patentable for at least the reasons for which the claims on which they depend are patentable.

Canceled claims, if any, have been canceled without prejudice or disclaimer.

Any circumstance in which the applicant has (a) addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner, (b) made arguments for the patentability of some claims does not mean that there are not other good reasons for patentability of those claims and other claims, or (c) amended or canceled a claim does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

No fee is believed due. Please apply any other charges to deposit account 06-1050, referencing attorney docket 10559-305US1.

Respectfully submitted,

Date: May 2, 2006

/Ido Rabinovitch/
Ido Rabinovitch
Attorney for Intel Corporation
Reg. No. L0080

Fish & Richardson P.C.
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

21320975.doc